Using Active Data to Provide Smart Data Surveillance to E-Science Users

Anthony Simonet, Kyle Chard, Gilles Fedak, Ian Foster

Using Active Data to Provide Smart Data Surveillance to E-Science Users

Anthony Simonet*, Kyle Chard†, Gilles Fedak* and Ian Foster†
*Laboratoire de l'Informatique et du Parallélisme, Inria, University of Lyon, ENS Lyon, Lyon, France
Email: {anthony.simonet, gilles.fedak}@inria.fr
†Computation Institute, University of Chicago and Argonne National Laboratory, Illinois, USA
Email: {chard, foster}@uchicago.edu

***Abstract*—Modern scientific experiments often involve multiple storage and computing platforms, software tools, and analysis scripts. The resulting heterogeneous environments make data management operations challenging; the significant number of events and the absence of data integration makes it difficult to track data provenance, manage sophisticated analysis processes, and recover from unexpected situations. Current approaches often require costly human intervention and are inherently error prone. The difficulties inherent in managing and manipulating such large and highly distributed datasets also limits automated sharing and collaboration. We study a real world e-Science application involving terabytes of data, using three different analysis and storage platforms, and a number of applications and analysis processes. We demonstrate that using a specialized data life cycle and programming model—Active Data—we can easily implement global progress monitoring, and sharing; recover from unexpected events; and automate a range of tasks.**

I. Introduction

In virtually all scientific domains, from large scale simulations to sensor networks composed of scientific instruments, e-Sciences produce large volumes of often distributed data. Interacting with such large and growing data drives the development of new e-infrastructures and processing workflows which leverage techniques from parallel computing, system management, and scripting languages to glue various systems and tools together in ad-hoc ways. Thus, the difficulty of controlling such complex scientific data life cycles comes not only from the size of data but also from the overwhelming sum of human interaction required. Often, these aspects ultimately create bottlenecks in data-intensive science. Automating human tasks, recording provenance, efficiently monitoring progress and detecting the rare errors among the mass of things that go right are extremely difficult to achieve in the context of e-Science.

We study a materials science experiment at a large scale research facility as a representative e-Science use case, in which datasets are handled by several systems and software. We propose a smart surveillance framework that allows users to monitor data life cycle progress in real-time. This framework offers a unique combination of features: ability to monitor heterogeneous systems and distributed infrastructures, automated data tagging and ability to convey tags across systems, data filtering, and rule-based programming. We demonstrate the benefit of this data surveillance system by implementing a prototype based on the Active Data life cycle model [1].

Our contributions include: 1) modeling the entire end to end data life cycle in an experiment, with the ability to expose what happens *inside* the individual systems; 2) extensions to the Active Data model to support data tagging based on Petri Net colors and guarded transition-based code execution; and 3) a data surveillance framework that leverages the life cycle model and allows users to implement sophisticated features at a high level. We demonstrate the flexibility of our framework with some of these features: progress monitoring, automation of human tasks and recovery from unexpected failures.

The rest of this paper is structured as follows: Section II describes our e-Science use-case; Section III discusses the objectives of our surveillance framework and Section IV describes the framework implementation; Section V evaluates the framework with user applications; Section VI discusses related work and Section VII summarizes the paper.

II. Background

We study a real world e-Science application used by a materials science experiment at the Advanced Photon Source (APS), a user facility at Argonne National Laboratory. The experiment generates up to 1TB of data per day that must be moved, cataloged, and analyzed across several different systems.

A. APS Experiment

The materials science experiment focuses on analyzing different sample materials using synchrotron x-rays. Scientists apply techniques such as high-energy diffraction microscopy (HEDM) and combined high-energy small- and wide-angle x-ray scattering (HE-SAXS/WAXS) to characterize different samples. The end-to-end experiment process undertaken by the scientists, shown in Fig. 1, is both compute and data intensive, using thousands of cores to enable near-real-time analysis of data as it is acquired. Experiments at the beamline currently generate 3-5TB of data per week.

Data is obtained from a beamline detector with a direct connection to an acquisition machine with modest storage and compute resources. As data is acquired it is moved to a shared cluster with greater storage capacity. After transfer to the shared cluster data is processed with several data reduction and aggregation scripts. Files related to a particular experiment or sample are grouped into "datasets". The data is then cataloged in the Globus Catalog. Here automated python-based parsers are used to extract metadata related to the user, experiment and sample. Knowledge of the Nexus HDF format is used to extract

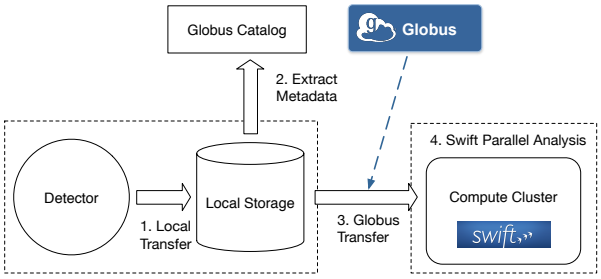

Fig. 1: The Advanced Photon Source experiment

structured metadata from the file. The catalog associates various different raw and derived data, scripts, and metadata into self-contained datasets. Following cataloging, data is moved to large scale compute resources. Here a parallel Swift-based analysis pipeline is run to fit a crystal structure to the observed image. This analysis is computationally intensive and involves iterative processing of many rows using a C function on each parameter and a reduction phase to merge the results into a single output. Throughout the analysis provenance information is recorded in the Catalog and associated with the sample dataset.

B. Experiment Tools

While not a complete list of tools used in this experiment, the following are the core tools that act upon data.

Globus provides high performance, secure, third party data transfer and synchronization. It is operated as Software-as-a-Service and is accessible via a web interface or REST API.

Globus Catalog is a service that enables the creation and management of user-defined "catalogs" containing datasets and references to associated data and metadata. Within a catalog, users can create datasets, associate files and directories, and specify user-defined metadata.

Swift is a parallel scripting language designed for composing applications into workflows that can be executed in parallel and distributed environments. Swift is implicitly parallel: users do not explicitly program their workflows to be parallel or to handle synchronization, file transfer, or execution locations.

C. Active Data

Active Data is a system that aims to make distributed data management more efficient. It offers a formal and graphical model to represent the life cycle of distributed data in terms of data state and state transitions, from creation to deletion. This model is able to represent what individual systems do with data internally and to *compose* such models that represent movement from one system to another, offering a high-level and *flat* view of large applications, while also abstracting hardware and software complexity. Based on this application model, Active Data offers an API and a *transition-based* programming model. Conceptually, data management systems report on their internal activity, and users specify actions to be conducted when specific data state transitions are reported, making it easy to implement decisions and optimizations.

III. Objectives

While the APS use case currently satisfies the needs of its users, there is potential for significant inefficiency, unreported failures and even errors due to the complexity of dealing with several terabytes of data and a number of different tools and systems. Here we present three useful features desired by scientists that appear simple at small scales and when executed on a single machine but present significant challenges with large distributed datasets.

1) *Progress Monitoring*: Monitoring is not limited to estimating completion time, but also: i) receiving a single relevant notification when several related events occurred in different systems; ii) quickly noticing that an operation failed within the mass of operations that completed normally; iii) identifying steps that take longer to run than usual, backtracking the chain of causality, and fixing the problem at runtime.

2) *Automation*: The APS experiment, like many scientific experiments and workflows, requires explicit human intervention to progress between stages and to recover from unexpected events. Such interventions cannot be easily integrated in a traditional workflow system, because they reside at a level of abstraction above the workflow system.

3) *Error Discovery and Recovery*: Each system participating in the APS experiment has only a partial picture of the entire process, which impairs the ability to recover from unexpected events. Thus, when such events occur, systems often fail ungracefully, leaving the scientists as the only one able to resolve the problem through costly manipulations. We aim to automate low level and error prone human interventions that delay experiment completion and places a burden on scientists.

IV. System Design

We next present the data surveillance framework designed to satisfy the objectives presented above.

A. Data surveillance framework

The data surveillance framework provides several unique features: it is able to track files, datasets, and elements related to data (such as file transfers and metadata); it receives events from different systems and integrates the identifiers of data and related elements in a single global namespace that allows advanced querying and filtering; data replicas and their current state are exposed as *tokens* with a unified identifier used across systems. Tokens provide a model for linking together related data in different systems; finally it allows users to be notified of the progress of their data and to automatically run custom code at many operational stages of the life cycle.

B. Active Data

Active Data provides a non-invasive method of improving coordination and obtaining feedback from loosely coupled systems. Specifically, we use Active Data's life cycle model to individually represent and expose the internal life cycle of the three systems used in the APS experiment: Globus, Globus Catalog and Swift; and Active Data's composition ability to represent how data moves between them.

In Active Data, the model of a system's life cycle is composed of places, represented by circles and transitions; places represent all the possible states of data; transitions represent all the legal changes of states. A set of directed arcs connect places to transitions, and transitions to places. Places may contain one or several tokens; each token represents a data replica, and the current place represents the current state of the replica. Several tokens on different places represent several replicas in different states at the same time. Each token holds an identifier that links the token to the actual piece of data (a URL or a path, for example). In addition, Active Data offers the ability to automatically run code to react to transitions being triggered. This code can be run anywhere (not necessarily where the event occurred) and can access the complete state of the life cycle.

C. APS Experiment Life Cycle Model

We now present the life cycle model of data in the APS experiment. We use the notation *A.B* to mean transition *B* in system *A*.

The complete model, shown in Fig. 2, is divided into six parts, separated by *composition transitions* (in red). On the left, we represent the detector with a minimal life cycle, that is a CREATED and a TERMINATED place (the only two places that are required in any life cycle model), with a transition between them; The detector's CREATED place is connected to the life cycle of a Globus transfer with a composition transition. This special transition represents the moment when Active Data records the identifier in the destination system (Globus) and maps it to the identifier in the source system (the detector).

In the Globus model, a token represents a transfer task containing a directory tree with several files. A transfer task can succeed or fail. In the case of success, the composition transition creates a new token in the shared storage.

The shared storage is also a minimal life cycle. From here, two things happen: a Python script registers each directory in the completed transfer as a dataset in the Globus Catalog and annotates it with metadata extracted from the files; and another Globus transfer copies each dataset to a computing platform for analysis. The order in which these two stages occur is not important in the model; the order in which both transitions are triggered will simply indicate which happened first.

On the Globus Catalog model, metadata can be added to a dataset, and all metadata can be removed at once. On the Globus transfer model, a transfer can succeed or fail. If the transfer succeeds, the *Start Swift* composition transition can be executed for each file in the dataset. When the composition transition is executed, it maps the Globus task identifier to the identifier of a file from the dataset (its path in the computing cluster). These files are used as input to the Swift script, that will in turn produce an output file. Swift features a *self composition*: each input file is naturally linked to the output file it produced by the *Derive* transition. An error in the Swift script results in the output file not being created and the *Failure* transition being triggered.

In addition to the CREATED place, each independent life cycle contains a TERMINATED life cycle that is used to indicate to Active Data that a life cycle is complete and that the associated resources can be freed.

D. Event Capture

Having represented the entire model with Active Data, information from an execution of the APS experiment has to be mapped to the model. This translates to creating a token in the model for every file, transfer and dataset manipulated, and then informing Active Data of *transitions*, i.e., when a system alters the state of a file, transfer or dataset. In Active Data, this is called "publishing transitions." We use several different strategies to instrument these systems to publish Active Data transitions. The methods for instrumenting the three systems—Globus transfers, Globus Catalog and Swift—are completely non-intrusive and reusable. Depending on the system's design, we either query updates from its API or derive data transitions from log files. Only user level scripts have been modified to publish transitions after some tasks are launched.

The individual life cycle models and the scripts we have developed can be used independently from the rest of this work by scientists wanting to make their tools and workflows "Active Data enabled."

E. Active Data extensions

The APS experiment generates Active Data transitions and user notifications. To ensure notifications remain manageable, we have implemented two new features in Active Data:

1) *Token Tags*: To achieve our goal of discriminating tokens, we borrow from Colored Petri Nets and implement tags (like colors) that can be attached to tokens. We extend the token API in Active Data to provide users with the possibility to add and remove tags, and to test if a particular tag is attached to a token. When a composition transition is published, tags from the input token are copied to the output tokens, allowing external information to travel through all systems.

2) *Handler Guards*: With token tags in place, we again borrow from Colored Petri Nets by implementing guards that can be attached to user subscriptions. When subscribing to a transition, a user supplies a `HandlerGuard` object that implements a single `boolean` method. The user is notified of a given event if the method returns `true` against the corresponding $(transition, token)$ couple.

F. Tagging

To make tokens more useful to users, we attach relevant information to them using our extended tag features. Tags are automatically placed by Active Data when transitions are executed; Table I describes the tags used in the APS experiment.

TABLE I: APS life cycle transitions and corresponding tags.

Transition	Tags
Detector.Start transfer	Detector name
Globus transfer.End transfer	Dataset name
Shared storage.Start transfer	File type
Globus transfer.Start Swift	"swift-input"
Swift.Initialize	Program name
Swift.Derive	"swift-output"

V. RESULTS

To evaluate our approach we demonstrate the ease by which the four objectives described in section III are provided using our proposed surveillance framework.

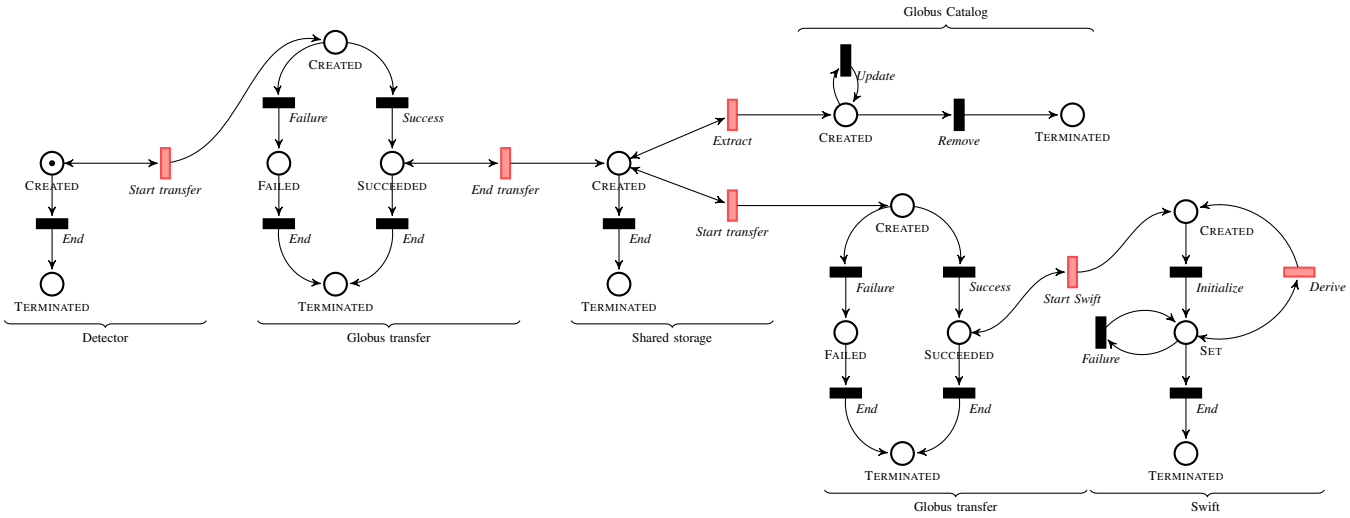

Fig. 2: The life cycle model for the APS experiment comprises three main systems (Globus transfers, Globus Catalog, and Swift) as well as intermediate storage. Data and information transfers between systems are represented with red *composition transitions*.

A. Monitoring Progress

To monitor progress by observing all systems in the experiment at the same time, we have developed a sample user code that records (in a log file) when the experiment lifecycle is complete, that is, when a dataset from the detector is fully processed by Swift. In order to generate this record, we need to know how many files are contained in each dataset, what dataset a file belongs to, and how many files from each dataset Swift has processed. This user code is configured to run when files are transferred from the detector (transition *Detector.Start transfer*) and when Swift derives an output file from an input file (transition *Swift.Derive*).

We present pseudocode for this sample in Algorithm 1. The code keeps two counters for each dataset, recording how many files it contains, and how many have been processed. If the event that triggered the execution was a dataset leaving the detector, then we store how many files it contains. If the execution was triggered because Swift finished processing a file, we update the counter of processed files for the dataset, despite the fact that Swift has no knowledge of which dataset the file originated from. The code learns this information by querying the Active Data API. Finally, if the last file in the dataset has been processed, the log file is updated.

Algorithm 1 Monitor the progress of the experiment life cycle.

```
ad ← ActiveDataClient.getInstance()
datasetSize ← []
datasetTreated ← []
function HANDLER(transition, isLocal, inToken, outToken)
    dsId ← ""
                                  [Count total number of files in the dataset]
    if transition = "Detector.Start transfer" then
        dsId ← inToken.getUid()
        size ← len(listFiles(dsId))
        datasetSize[dsId] ← size
                    [Count number of treated files in the dataset so far]
    if transition = "Swift.Derive" then
        lc = ad.getLifeCycle(inToken)
        createdTokens ← lc.getTokens("Detector.Created")
        dsId = createdTokens[0].getUid()
        datasetTreated[dsId] += 1
                                                    [If done, update file]
    if datasetSize[dsId] = datasetTreated[dsId] then
        out ← open("aps.log")
        out.write("Done: " + dsId)
        out.close()
                                  [Subscribe the handler to both transitions]
ad.subscribeTo("Detector.Start transfer", handler)
ad.subscribeTo("Swift.Derive", handler)
```

B. Automation

We now demonstrate that tasks that were previously performed manually can be easily automated using our approach. In particular, we discuss the automation of two tasks.

First, when a transfer to the shared storage completes, we automatically run the Python metadata extraction script. A user code is set to run after transition *Globus Transfer.End transfer* is triggered; this code examines the token produced by the transition and uses its identifier to run the Python extraction script directly.

Second, when the transfer of a HDF file to the compute cluster completes, we execute a specific Swift analysis script with a user code that is set to run after transition *Globus transfer.Start Swift* is triggered. This code is similar to the first example, except that we restrict its execution with a guard. Because tokens have been tagged according to their corresponding file type, we can set the user code to be run only for tokens that point to an HDF file.

C. Error Detection and Recovery

We finally consider the problem of detecting and recovering from experiment-wide errors. On occasion faulty files acquired by the detector are only identified during analysis, near the end of the process. In this situation, the measures taken by users are to drop the dataset entirely and to reacquire the dataset with the same (or fixed) parameters. This procedure

can take a considerable amount of time and represents a significant overhead on the experiment process. However, the surveillance framework can detect these errors immediately by observing Swift failures and automatically take appropriate recovery measures, in addition to notifying the user.

The user code benefits from the high-level view of the whole process to automatically recover from errors in the same way that the user would. It is executed on a node of the shared storage cluster for any Swift token with a "failure-corrupted" tag. The code uses the Active Data client API to retrieve the representation of the life cycle of the faulty file, as previously shown in section V-A. The representation is used to access remote elements through their identifier. The code removes associated metadata from the Globus Catalog (which will trigger the *Globus Catalog.Remove* transition), removes the files from the shared storage, triggers *Detector.End* for the dataset and finally notifies the user via email. Finally, user code running on the detector's acquisition machine can react to the *Detector.End* transition being triggered by running the acquisition again.

VI. Related Work

Scientific workflow systems such as Taverna [2], Galaxy [3], and LONI [4] have been used in many domains including bioinformatics, genomics, astronomy, and medical imaging. These systems allow researchers to orchestrate a series of tools and services into a cohesive workflow and the workflow system handles the flow of data between tools. In contrast to Active Data, workflow users must manually "wrap" tools and services before they can be used in a workflow. Many workflow systems have been extended to capture provenance [5] and other systems, such as the Provenance Aware Storage System [6] capture system events at the storage level so that operations on files are captured. While such approaches can be used for tracking actions within and across tools, Active Data supports a model capable of making sense of interlinked information derived from participating systems. Error detection and recovery methods range, from passive methods, such as error logging, which allow to discover (not recover) errors, to statistical analysis and machine learning techniques that can predict and detect faults during workflow executions [7]. These models allow actions to be taken based on events within the workflow; Active Data, on the other hand, focuses on a wider scope of errors including user errors and events occurring outside the workflow ("user-steering" [8]).

VII. Conclusion

Large scientific experiments introduce new data management challenges and necessitate the development of novel techniques to manage the entire data life cycle.

We have proposed an implicit surveillance approach based on Active Data which allows users to monitor real-time data life cycle progress non-invasively. It provides other important features, including automation, progress monitoring and error detection and recovery. We demonstrated the ease through which users and developers alike can leverage such functionality by instrumenting a real-world materials science experiment at a large user facility. The proposed approach is both efficient and easy to use as users need not perform extensive application-specific development. Instead they leverage instrumented existing and commonly used scientific tools, while one-off investments to make scientific tools "Active Data enabled" benefit all users.

In future work, we aim to apply this same set of instrumented tools to other scientific experiments in other domains. We will also continue to instrument commonly used tools such as Hadoop to meet the requirements of an increasingly large user community.

References

[1] A. Simonet, G. Fedak, M. Ripeanu, and S. Al-Kiswany, "Active data: a data-centric approach to data life-cycle management," in *Proceedings of the 8th Parallel Data Storage Workshop*. ACM, 2013, pp. 39–44.

[2] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. de la Hidalga, M. Vargas, S. Sufi1, and C. Goble, "The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 4, no. W1, pp. W557–W561, 2013.

[3] J. Goecks, A. Nekrutenko, J. Taylor, and The Galaxy Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biology*, vol. 11, no. 8, p. R86, 2010.

[4] D. Rex, J. Ma, and A. Toga, "The LONI pipeline processing environment," *NeuroImage*, vol. 19, no. 3, pp. 1033–1048, 2003.

[5] S. B. Davidson and J. Freire, "Provenance and scientific workflows: Challenges and opportunities," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1345–1350.

[6] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proceedings of the 2006 USENIX Annual Technical Conference*, 2006, pp. 43–56.

[7] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Juve, G. Mehta, F. Silva, and K. Vahi, "Online fault and anomaly detection for large-scale scientific workflows," in *2011 IEEE 13th International Conference on High Performance Computing and Communications (HPrC)*, Sept 2011, pp. 373–381.

[8] M. Mattoso, K. O. na, F. Horta, J. Dias, E. Ogasawara, V. Silva, D. de Oliveira, F. Costa, and IgorAraújo, "User-steering of hpc workflows: State-of-the-art and future directions," in *Proceedings of the 2Nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, ser. SWEET '13. ACM, 2013, pp. 4:1–4:6.